CONTROL DATA CORPORATION

DOCUMENT CONTROL FORM

o DOCUMENT TYPE: IDS

o LOG ID: ARH7016

o TITLE: DoD Internet Protocol

o ABSTRACT: Describes the implementation of the DoD Internet Protocol as Layer 3B in the CDCNET environment.

o PRODUCT AFFECTED: CDCNET

o AUTHOR: John R. Lyman III

o MAIL STATION: ARH207

o EXTENSION: 3087

o PROJECT: Telnet/IP/TCP - CSU Document

o SECTION: ACSD

IDS

o RSM or PSR #

o SIP Number(s)

o PUB #

o PRM #

o Redesign* o Reimplementation*

Name	Approval Initials	Date
o Internal Reviewer*		
o Project Leader		
o Unit Manager T.C. McGee	TCM	9/25/85
o Section Manager		
Design Team Referee B.S. Sekhon	BSS	9/25/85

DESIGN TEAM: CDCNET DEFAULT CYCLE: 0 WORKING DAYS

o Special Distribution: D.R. WALLACE ARH207

Referee's Distribution Codes: ACTS-B, AC-B

PAGE 2

TO BE FILLED OUT BY DCS

DCS LOG ID ARH7016

NUMBER	DATE	TYPE	SUBMITTED BY	DEFAULT APPROVAL DATE	CURRENT STATUS			
					REVIEW	HOLD	APPRVD	WITH-DRAWN
1	9-26-85	ORIG	J. R. Lyman	NONE			A	

DOD Internet Protocol

Internal Design Specification

September 24, 1985

AUTHOR: _____
John R. Lyman III

TDRB APPROVAL: _____

AD&C APPROVAL: _____

PROPRIETARY NOTICE

The technical content set forth in this document is the property of Control Data and is not to be disseminated, distributed or otherwise conveyed to third parties without the express written permission of Control Data.

This document is to be used for planning purposes only. It does not include any explicit or implied commitments for any specific release dates or feature content. These matters are currently under active development, and as such are subject to revision and replanning as circumstances warrant.

RECORD OF REVISION

Revision	Description	Author	Date
01	Draft Version	JRL3	07/30/85
02	Updated after IP walkthrough	JRL3	09/24/85

1.0 INTRODUCTION

This document describes the internal design of the Internet Protocol (IP) module. The IP module implements the full DoD IP protocol, and adds some minor features not in the DoD specification. The IP module includes both the IP and the ICMP protocols. Due to the fact that the DI hardware is capable of connecting to many networks at the same time, the IP module extends the DoD specification by providing a datagram forwarding service between any number of connected IP networks. Routing decisions are not made by the IP module, instead the IP static routing module makes all necessary routing decisions.

In the CDCNET environment the IP module will use the 3A intranet module. This allows the IP protocol to be active on any of the network solutions that the DI software and hardware provide.

2.0 REFERENCES

2.0 REFERENCES

The following manuals contain material that either defines the operations of the IP module and the modules it interfaces to, or provides additional insight into the use of the IP module.

[1]	RFC-791	SRI	Internet Protocol
[2]	RFC-792	SRI	Internet Control Message Protocol
[3]	MIL-STD-1777	DoD	Internet Protocol Standard
[4]	ARH6265	CDC	DoD Internet Protocol ERS
[5]	ARH6879	CDC	3A Intranet ERS
[6]	JRL3...	CDC	DoD IP Static Routing ERS

3.0 ENVIRONMENT

3.1 HARDWARE

The IP module has no special hardware requirements. The IP module is part of the CDCNET software and will run on a 68000 based Device Interface (DI). The module will be written in the CYBIL language, and will be compiled and bound using SES tools on a CYBER mainframe.

3.2 SOFTWARE

The IP module depends on a number of other software components in order to function in the DI. The following sections list these components and itemize the services of each component that the IP module uses.

3.2.1 3A INTRANET MODULE

The 3A Internet module provides a basic point-to-point data transfer service between two systems connected to a common network solution. The interface between the IP module and the 3A module will require that the following services be provided by the two modules.

1. The 3A module will provide an open SAP service such that the IP module is able to open a SAP through which it will receive datagrams from all IP type network solutions.

2. The 3A module will provide a routine which the IP module will call to send datagrams to other connected systems.

3. The IP module will provide a routine that the 3A module will call to present datagrams that are received on IP network solutions.

4. The IP module will provide a routine that the 3A module will call to present status indications for all IP network solutions. This status will include the maximum datagram size for each network.

3.2.2 IP ROUTING MODULE

The IP module does not make any routing decisions, for each datagram that it receives from an upper layer module or the 3A module, the IP routing module is called to determine the next destination of the datagram. The IP routing module provides a routine which uses the source address, the destination address, and the IP options of the datagram to determine the next destination that the datagram should be sent to. This routine will then return the source address, destination address, the type of destination, and the maximum datagram size to the IP module The IP module can then route the datagram to the proper location.

3.2.3 STATISTICS MANAGER

The IP module will provide a report procedure that the statistics manager can call to display the statistics that the IP module has collected. A statistics SAP will be opened during initialization, and the required pointers will provided to the statistics manager.

3.2.4 EXECUTIVE COMMON ROUTINES

The IP module will use a number of the common subroutines that are provided as part of the executive. The following services will be expected from the executive.

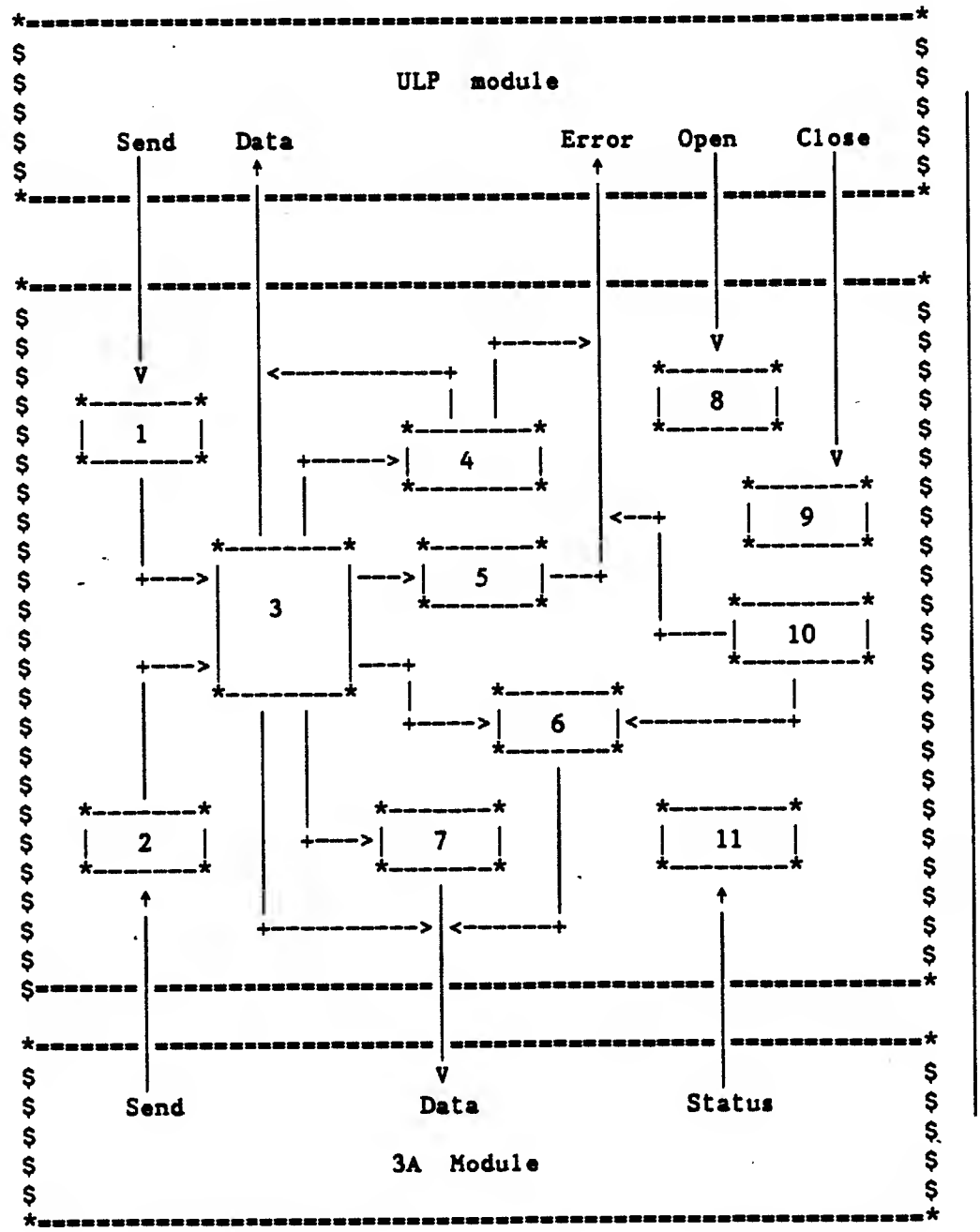
1. Timer interrupts.
2. Buffer creation.
3. Buffer destruction.
4. Addition of data to buffers.
5. Removal of data from buffers.
6. Intertask messages.

4.0 DESIGN OVERVIEW

The IP module provides the ability to transmit datagrams throughout a packet-switched internetwork. This service is provided as a number of direct subroutine calls. The IP module will not run as a task in the DI due the simplicity of the processing that it does. Each task that uses a service of the IP module will provide the cpu time that the subroutines in the IP module need to provide that service. The following page contains a diagram of the flow of execution thru the IP module. The subroutine names listed below correspond to the numbers in the diagram.

1. Receive_ULP_Data
2. Receive_3A_Data
3. Process_IP_Datagram
4. Process_Fragment
5. Process_ICMP_Datagram
6. Generate_ICMP_Datagram
7. Generate_Fragment
8. Open_SAP_Processor
9. Close_SAP_Processor
10. Process_Timer_event
11. Receive_3A_Status

4.0 DESIGN OVERVIEW



4.1 FUNCTIONAL STRUCTURE

4.1.1 CLOSE_SAP_PROCESSOR

The close_SAP routine is called by the ULP to close an open SAP. This will clear the data structure and allow another module to use the released protocol number. This routine should be called by the ULP as soon as it is done transmitting datagrams with the particular protocol number.

Call format

```
PROCEDURE ip_close_sap (
   protocol   : 0..255;
   sapid      : INTEGER;
   VAR status : ip_status_type);
```

protocol	In	This value specifies the user protocol to the IP module.
sapid	In	This is the SAPid returned on the original open request.
status	Out	This is the status of the request. The following values may be returned: ip_successful ip_sap_not_open

Global data accessed

1. The Protocol Status Table (PST) is updated and the PST record is deleted.
2. Any Reassembly Buffers in use are released.
3. The Statistics Data Structure (SDS) is updated.

4.0 DESIGN OVERVIEW
4.1.1 CLOSE_SAP_PROCESSOR

General Algorithm

```
BEGIN
   IF the SAPID is in use THEN
      Clear the PST record pointer;
      Release any reassembly buffers;
      Release the PST record;
      IF (ten or more unused protocols) THEN
         Reduce the array size;
      IFEND;
      Update the SDS counters;
      Force statistics for the protocol;
      Release the statistics record for the protocol;
      RETURN (ip_successful)
   ELSE
      RETURN (ip_sap_not_open)
   IFEND
END.
```

4.1.2 GENERATE_ICMP_DATAGRAM

This routine is called by the process_ip_datagram routine in order to generate a ICMP datagram. The appropriate information will be passed in to describe the type of error that has been detected. The datagram will be created and sent to the sender of the datagram in error.

Call format

```
PROCEDURE generate_icmp_datagram (
   header      : ip_header;
   options     : ip_option_rec;
   error_type  : icmp_status_type;
   error_code  : INTEGER;
   error_ptr   : INTEGER;
   new_address : ip_address;
   data        : buf_ptr);
```

header	In	This is a record which contains all of the information that was contained in the IP header of the datagram that is in error.
options	In	This is a byte array which contains the options that the IP header included.
error_type	In	This is the type of ICMP datagram that should be sent. The following types of datagrams may be sent. icmp_echo_reply icmp_dest_unreachable icmp_source_quench icmp_redirect icmp_echo_request icmp_time_exceeded icmp_parameter_error icmp_time_request icmp_time_reply icmp_info_request icmp_info_reply
error_code	In	This is the code that is passed in the datagram and determines the

		specific error with the type. The value of this parameter is dependent on the type parameter.
error_ptr	In	This is a pointer to the byte in error for the case of parameter errors.
new_address	In	This is the new address for a redirect datagram.
data	In	This is a pointer to the data to be sent with the ICMP datagram. For most error types the data buffer will contain the data portion of the datagram in error, only the internet header and the first 64 data bytes will be used. In the case of Echo, Timestamp, and information datagrams, the data buffer will contain all data comprising the ICMP datagram except for the first word which contains the type, code, and checksum.

Global data accessed

1. None.

General Algorithm

```
BEGIN
   Build the IP header.
   Compute the IP checksum.
   Build first word of datagram.
   Build remainder of datagram (depends on error_type).
   Compute the ICMP checksum.
   Send the datagram.
END
```

4.1.3 GENERATE_IP_FRAGMENTS

This routine is called by the process_ip_datagram routine in order to fragment a datagram that is too large to be transmitted on the appropriate network. The datagram will be divided into a number of fragments and each fragment will be sent out through the 3A module.

Call format

```
PROCEDURE generate_ip_fragments (
   header     : ip_header;
   options    : ip_option_rec;
   data       : buf_ptr;
   max_size   : INTEGER;
   network_id : net_id_type;
   system_id  : sys_id_type);
```

header	In	This is a record which contains the header of the datagram that needs to be fragmented.
options	In	This is a byte array which contains the options that the IP header should include.
data	In	This is a pointer to the data to be sent with the IP datagram.
max_size	In	This is the maximum number of bytes that each datagram is allowed to contain.
network_id	In	This is the 3A identifier for the network that the datagram is being sent to.
system_id	In	This is the 3A identifier for the the specific host that the datagram is intended for.

Global data accessed

1. None.

General Algorithm

```
BEGIN
   IF NOT header.dont_frag THEN
      dmax := max_size - options_size - 20;
      dmax := dmax - (dmax MOD 8);
      fcount := 0;
      WHILE data<>NIL DO
         Remove dmax bytes from the data buffer.
         header.offset := fcount;
         header.more_frags := (data<>NIL);
         IF fcount=0 THEN
            Build the fragment with all options.
         ELSE
            Build the fragment with repeat options only.
         IFEND;
         Send the fragment through 3A.
         fcount := fcount + dmax;
      WHILEND;
   ELSE
      Return(ip_fragmentation_needed);
   IFEND;
END
```

4.1.4 INITIALIZE_IP_MODULE

This routine is called to initialize the IP module. The routine sets up all of the data structures, opens a SAP with the 3A module, and opens a SAP with the statistics processor.

Call format

```
PROCEDURE initialize_ip_module (
   VAR status : INTEGER);
```

status	OUT	This is the status of the initialization request returned to the caller. This may be one of the following values. ip_successful ip_insufficient_resources

Global data accessed

1. All pointers in the Protocol Status Table will be set to NIL.
2. All counts in the Statistics Data Structure will be set to zero.
3. The timer list will be created.

General Algorithm

```
BEGIN
   Initialize the Protocol status table.
   Initialize the Statistics Data Structure.
   Create the timer list.
   Open a SAP with the statistics processor.
   Open an IPnet SAP with 3A.
END
```

4.1.5 OPEN_SAP_PROCESSOR

The open_SAP routine is called by the ULP module in order to open a SAP with the IP module. A open SAP allows the ULP module to use a specific protocol number and thereby send datagrams out on the network. Each SAP is identified by the SAPID that is assigned by the IP module, all communication between the modules will use this number for identification.

Call format

```
PROCEDURE ip_open_sap (
   protocol     : 0..255;
   data_ind     : ip_data_ind;
   error_ind    : ip_error_ind;
   VAR send_req : ip_send_req;
   VAR sapid    : INTEGER;
   VAR status   : ip_status_type);
```

protocol	In	This value specifies the user protocol to the IP module. The IP module will only allow one SAP for each protocol number. The ULP must specify an appropriate number between zero and 255.
data_ind	IN	This is a pointer to a user supplied routine, which the IP module will call to present data messages to the ULP.
error_ind	IN	This is a pointer to a user supplied routine, which the IP module will call to present error messages to the ULP.
send_req	OUT	This is a pointer to the IP modules routine to send data.
sapid	OUT	This is a 32 bit value that identifies the particular IP SAP in all later requests.
status	OUT	This is the status of the request. The following values may

be returned:
ip_successful
ip_protocol_inuse
ip_protocol_illegal
ip_insufficient_resources

Global data accessed

1. The Protocol Status Table (PST) is updated and a PST record is created.

2. The Statistics Data Structure (SDS) is updated.

General Algorithm

```
BEGIN
   IF first open_sap call THEN
      Call the initialization routine.
   IFEND
   IF legal protocol number THEN
      IF PST array too small THEN
         Enlarge  the PST array.
      IFEND;
      IF protocol number not in use THEN
         Create a PST record.
         Update the PST record pointer.
         RETURN (ip_successful);
      ELSE
         RETURN (ip_protocol_inuse)
      IFEND
   ELSE
      RETURN (ip_protocol_illegal)
   IFEND
END
```

4.1.6 PROCESS_IP_DATAGRAM

This routine will receive an IP datagram that has come from the ULP or 3A module. It will determine where the datagram is going, process reassembly, and do fragmentation.

Call format

```
PROCEDURE process_ip_datagram (
   header      : ip_header_rec;
   source      : ip_address;
   destination : ip_address;
   options     : ip_option_rec;
   data        : buf_ptr;
   VAR status  : ip_status_type);
```

header	In	This is a record which contains the IP header for the datagram.
source	In	This is the address of the sender. This address may be partially of completely unspecified if the datagram can from the ULP.
destination	In	This is the address of the remote IP that the data is being sent to.
options	In	This is an array which contains the option parameter data.
data	In	This is a pointer to a system buffer containing the data portion of the datagram.
status	Out	This is the status of the request. The following values may be returned: ip_successful ip_net_unreachable ip_host_unreachable ip_fragmentation_needed ip_option_error ip_sap_not_open ip_source_illegal

ip_destination_illegal
ip_protocol_illegal

Global data accessed

1. None.

General Algorithm

```
BEGIN
   Process the options;
   IF no option errors THEN
      CALL the routing module;
      CASE route OF
         =ULP=
            IF fragment THEN
               CALL process_ip_fragment;
            ELSE
               IF ICMP datagram THEN
                  CALL process_icmp_datagram;
               ELSE
                  Present the data to the ULP;
               IFEND;
            IFEND;
         =3A=
            Build datagram and send it out through 3A;
         =No route=
            RETURN (error_code_from_routing);
      CASEND;
   ELSE
      RETURN (ip_option_error);
   IFEND;
END
```

4.1.7 PROCESS_ICMP_DATAGRAM

This routine will receive ICMP datagrams from the process_ip_datagram routine. The routine will generate an error indication to the ULP or generate an ICMP datagram and send it through 3A.

Call format

```
PROCEDURE process_icmp_datagram (
   source      : ip_address;
   destination : ip_address;
   data        : buf_ptr);
```

source	In	This is the address that the datagram originated from.
destination	In	This is the address that the datagram was sent to.
data	In	This is a pointer to a system buffer which contains the data that was received in the datagram.

Global data accessed

1. None.

4.0 DESIGN OVERVIEW
4.1.7 PROCESS_ICMP_DATAGRAM

General Algorithm

```
BEGIN
   Pull the ICMP header out of the data buffer.
   CASE error_type OF
      =icmp_echo_reply=
         Do nothing.
      =icmp_time_reply=
         Do nothing.
      =icmp_info_reply=
         Do nothing.
      =icmp_dest_unreachable=
         CASE error_code OF
            0:  ULP_error_ind (ip_net_unreachable);
            1:  ULP_error_ind (ip_host_unreachable);
            2:  ULP_error_ind (ip_protocol_unreachable);
            3:  ULP_error_ind (ip_port_unreachable);
            4:  ULP_error_ind (ip_fragmentation_needed);
            5:  ULP_error_ind (ip_route_failed);
         CASEND;
      =icmp_source_quench=
         ULP_error_ind (ip_congestion);
      =icmp_redirect=
         Inform IP routing module;
      =icmp_echo_request=
         Strip the first 4 bytes from the data buffer;
         Generate_icmp_datagram (dest,src,0,0,0,0,data);
      =icmp_time_exceeded=
         CASE error_code OF
            0:  ULP_error_ind (ip_timeout);
            1:  ULP_error_ind (ip_assembly_timeout);
         CASEND;
      =icmp_parameter_error=
         ULP_error_ind (ip_option_error);
      =icmp_time_request=
         Strip the first 4 bytes from the data buffer;
         Update the timestamps in the data buffer;
         Generate_icmp_datagram (dest,src,14,0,0,0,data);
      =icmp_info_request=
         Strip the first 4 bytes from the data buffer;
         Generate_icmp_datagram (dest,src,16,0,0,0,data);
  CASEND;
END
```

4.1.8 PROCESS_IP_FRAGMENT

This routine is called by the process_ip_datagram routine to process a datagram fragment. If the fragment contains the correct header information then the fragment is added to the datagram buffer chain. If the datagram is completely assembled then the datagram will be presented to the user and the structure will be released.

Call format

```
PROCEDURE process_ip_fragment (
   header      : ip_header;
   source      : ip_address;
   destination : ip_address;
   options     : ip_option_rec;
   data        : buf_ptr);
```

header	In	This is a record containing the header information that came in the datagram.
source	In	This is the source address from the IP header.
destination	In	This is the destination address from the IP header.
options	In	This is an array of bytes which contains the options from the IP header.
data	IN	This is a pointer to a system buffer which contains the data that came in the datagram.

Global data accessed

1. This routine will access the protocol_status_table to find the address of the reassembly buffer.

2. The routine will then access the reassembly buffer, add the datagram to the buffer if appropriate, and release the buffer when the datagram is completed and presented to the ULP.

3. The timer entry for the reassembly buffer will be updated.

General Algorithm

```
BEGIN
   IF a reassembly buffer exists for this protocol THEN
      IF header fields and options match THEN
         Find the correct place in the linked list.
         Insert the new data into the linked list.
         IF datagram completed THEN
            Deliver datagram to ULP.
            Release the buffer space.
            Release the timer entry.
         ELSE
            Update the timer entry.
         IFEND;
      ELSE
         Do nothing.
      IFEND;
   ELSE
      Create a reassembly buffer and store the data.
      Create a timer entry for the buffer.
   IFEND;
END
```

4.1.9 PROCESS_TIMER_EVENT

This routine is called by the executive to process the expiration of a system timer. The routine processes the timer list and releases any reassembly buffers whose timers have expired. The executive timer will expire periodically once a second.

Call format

```
PROCEDURE process_timer_event (
   parameter : ↑CELL);
```

parameter	In	This is the parameter that was passed to the executive when the timer was initiated.

Global data accessed

1. The the timer list will be accessed and expired timer records will be released.
2. The reassembly buffer of any timer record that is released will also be released.

General Algorithm

```
BEGIN
   cur_rec := timer_list↑.next_rec;
   cur_rec↑.delta := cur_rec↑.delta - 1;
   WHILE cur_rec↑.delta=0 DO
      Delete the buffer pointer to by cur_rec↑.buffer_ptr;
      timer_list↑.next_ptr := cur_rec↑.next_rec;
      Release the timer rec pointer to by cur_rec;
      cur_rec := timer_list↑.next_rec;
   WHILEND;
```

```
      END
```

4.1.10 RECEIVE_ULP_DATA

The receive_ulp_data routine is called by the ULP in order to send data out to some other host on an IP network. The address of this routine is given to the ULP when an open request is made.

Call format

```
PROCEDURE receive_ulp_data (
   header      : ip_header_rec;
   source      : ip_address;
   destination : ip_address;
   options     : ip_option_rec;
   sapid       : INTEGER;
   data        : buf_ptr;
   VAR status  : ip_status_type);
```

header	In	This is a record which contains the IP header for the datagram. The ULP module does not fill in the entire header only the user specified fields as noted in the ERS [4].
source	In	This is the address of the sender. This address may be partially of completely unspecified if the datagram can from the ULP.
destination	In	This is the address of the remote IP that the data is being sent to.
options	In	This is an array which contains the option parameter data.
sapid	In	This is the SAPid returned by the original open request.
data	In	This is a pointer to a system buffer containing the data portion of the datagram.
status	Out	This is the status of the

request. The following values may be returned:

```
ip_successful
ip_net_unreachable
ip_host_unreachable
ip_fragmentation_needed
ip_option_error
ip_sap_not_open
ip_source_illegal
ip_destination_illegal
ip_protocol_illegal
```

Global data accessed

1. The PST is accessed to validate the SAPID and to determine the protocol number.

2. The packet and byte counts in the SDS are updated.

General Algorithm

```
BEGIN
   IF valid SAPID and protocol THEN
      CALL process_ip_datagram (status);
      RETURN (status);
   IFEND;
END
```

4.1.11 RECEIVE_3A_DATA

This routine is called by the 3A module to present data indications. As each datagram is received from the network it will be delivered to the ULP, be added to the appropriate reassembly buffer, or be sent back to 3A. If the datagram is damaged or undeliverable then it will be discarded and an ICMP datagram will be sent back to the source.

Call format

```
PROCEDURE receive_3a_data (
   multicast     : BOOLEAN;
   receive_netid : net_id_type;
   sending_sysid : sys_id_type;
   VAR datagram  : buf_ptr);
```

multicast	In	This flag will be TRUE if the datagram was sent as a broadcast datagram.
receive_netid	In	This is the network identifier of the network solution that the datagram was received on.
sending_sysid	In	This the system identifier of the system that transmitted the datagram.
datagram	In	This is a pointer to the system buffer which contains the datagram.

Global data accessed

1. May access the protocol status table.

2. Will update the SDS counters.

4.0 DESIGN OVERVIEW
4.1.11 RECEIVE_3A_DATA

General Algorithm

```
BEGIN
   Pull the IP header out of the buffer;
   Unpack the source and destination addresses;
   CALL process_ip_datagram (status);
   IF status<>ip_successful THEN
      CALL generate_icmp_datagram;
   IFEND;
END
```

4.1.12 RECEIVE_3A_STATUS

This routine is called by the 3A module to present status indications. Since the IP module relies upon the IP routing module for routing decisions it does not need to know the status of networks, however, the statistics reporting requires that IP know what network solutions should be reported about. Therefore, this routine will check the active SDS buffer, if it receives an indication about a network that is not in the buffer it will add it.

Call format

```
PROCEDURE receive_3a_status (
   nib_ptr : nib_type);
```

nib_ptr	In	This is a pointer to the information block of the network whose status has changed.

Global data accessed

1. This routine will add networks to the active SDS buffer.

4.0 DESIGN OVERVIEW
4.1.12 RECEIVE_3A_STATUS

General Algorithm

```
BEGIN
   Search the active SDS buffer for the network id.
   IF (status is UP) AND (not found) THEN
      Add the network to the buffer.
   ELSE
      IF (status is DOWN) AND (found) THEN
         Force statistics for this network.
         Remove the record for this network.
      ELSE
         Do nothing.
      IFEND
   IFEND;
END
```

4.1.13 STATISTICS_PROCESSOR

Call format

```
PROCEDURE statistics_processor (
   sds_hdr    : ↑sds_header;
   function   : statistics_function_codes;
   reason     : statistics_reason_type;
   time       : report_time_type;
   param      : ↑cell;
   VAR status : statistics_function_status);
```

sds_hdr	In	This is a pointer to the header record of the statistics data structure.
function	In	This is the type of operation to be performed.
reason	In	This code indicates the reason that the statistics are being reported.
time	In	This is starting and ending time of the period that statistics should be reported for.
param	In	This is a pointer for use when the report is forced.
status	Out	This is the status returned to the caller.

Global data accessed

1. This routine will read and write the SDS tables.

General Algorithm

```
BEGIN
   CASE function OF
      =issue_report_and_clear_buffers=
         Build a log message from the inactive buffer.
         Send the log message.
         Clear the inactive buffer.
      =clear_buffers=
         Clear the inactive buffer.
      =start_collecting=
         Do nothing.
      =stop_collecting=
         Do nothing.
      =select_buffer1=
         IF local pointer set to buffer 2 THEN
            Copy the network table to SDS buffer 1.
            Set the local pointer to buffer 1.
         IFEND
      =select_buffer2=
         IF local pointer set to buffer 1 THEN
            Copy the network table to SDS buffer 2.
            Set the local pointer to buffer 2.
         IFEND
   CASEND;
END
```

4.2 DATA STRUCTURES

The ICMP datagrams use the following set of codes for the error code field that they contain.

```
CONST
   icmp_echo_reply = 0,
   icmp_dest_unreachable = 3,
   icmp_source_quench = 4,
   icmp_redirect = 5,
   icmp_echo_request = 8,
   icmp_time_exceeded = 11,
   icmp_parameter_error = 12,
   icmp_time_request = 13,
   icmp_time_reply = 14,
   icmp_info_request = 15,
   icmp_info_reply =16;

TYPE
   icmp_status_type = icmp_echo_reply..icmp_info_reply;
```

4.2.1 PROTOCOL_STATUS_TABLE

The protocol status table (PST) is used to keep track of the current condition of each protocol number. The PST is an array of pointers, where each active protocol number has a pointer to a record which contains the current status of the user of that protocol number. If a particular protocol number is not in use then the pointer will be NIL and no PST record will exist. The size of the PST will increase and decrease dynamically as SAPs are opened and closed. An active PST record may point to a chain of reassemble buffers.

CYBIL data definitions

```
TYPE
   pst_array = ARRAY [*] OF ↑pst_rec;

   pst_rec = RECORD
      sapid     : INTEGER,
      protocol  : 0..255,
      prev_ptr  : ↑pst_rec,
      next_ptr  : ↑pst_rec,
      data_ind  : ip_data_ind,
      error_ind : ip_data_ind,
      rbuf_ptr  : ↑reassembly_buffer,
      sds_ptr   : ↑ip_protocol_rec,
   RECEND;

VAR
   pst_size : INTEGER;
```

Creation/Modification

1. The base array of the PST is dynamically allocated by the open and close SAP routines. On the first call to the open SAP routine the array will be created.

2. As each SAP is opened by the ULP, the open_sap routine will create a PST record containing all of the received information. The pointer in the base array will then be updated to point to the new record. IF necessary the

PST array will be enlarged.

3. When a SAP is closed by the ULP, the close_sap routine will set the pointer in the base array to NIL and release the PST record. All buffers in use for reassembly will also be released. If there are more than ten pointers at the end of the PST array that are not active, then the size of the array will be decreased.

4. Many of the IP routines will use the information in the table but the only routines that modify the table will be the open and close SAP routines.

4.2.2 REASSEMBLY_BUFFER

Each protocol will have a chain of reassembly buffers. Each buffer is constructed as a linked list with a header record and a list of smaller records that identify each piece of data in the buffer. The data which makes up each segment of the datagram being reassembled will remain in the system buffer that it was received in, however, it will be appended to buffers containing adjoining data. When the buffer is completed it will be contained in a single buffer. Each reassembly buffer will have an entry in the timer list which is used to limit the time used in the reassembly process.

CYBIL data definitions

```
TYPE
   reassembly_buffer = RECORD
      next_buffer : ↑reassembly_buffer,
      first_rec   : ↑buffer_rec,
      timer_ptr   : ↑timer_rec,
      source      : ip_address,
      destination : ip_address,
      identifier  : 0..0FFFF(16),
      precedence  : 0..7,
      security    : ip_security,
      first_frag  : BOOLEAN,
      last_frag   : BOOLEAN,
   RECEND,

   BUFFER_REC = RECORD
      first_group : INTEGER,
      last_group  : INTEGER,
      next_rec    : ↑reassembly_buffer,
      data        : buf_ptr,
   RECEND;
```

Creation/Modification

1. Each reassembly buffer will be created, maintained, and released by the routine that reassembles datagrams.

2. A single reassembly buffer may be released by the timer routine if the reassembly time expires.

4.2.3 STATISTICS_DATA_STRUCTURE

The statistics compiled by the IP module are stored in the structure described in this section. The data is updated by a number of routines throughout the IP module and the overall reporting and control is done by the statistics processor.

CYBIL data definitions

```
TYPE
   ip_sds_buffer = RECORD
      open_count    : four_byte_statistic_record,
      close_count   : four_byte_statistic_record,
      protocol_list : ↑ip_protocol_rec,
      network_list  : ↑ip_network_rec,
   RECEND,

   ip_protocol_rec = RECORD
      bytes_sent         : four_byte_statistic_record,
      datagrams_sent     : four_byte_statistic_record,
      bytes_received     : four_byte_statistic_record,
      datagrams_received : four_byte_statistic_record,
      resource_errors    : four_byte_statistic_record,
      content_errors     : four_byte_statistic_record,
      next_rec           : ↑ip_protocol_rec,
   RECEND,

   ip_network_rec = RECORD
      network_id   : network_id_type,
      network_name : ↑CELL,
      local        : ARRAY [0..5] OF four_byte_statistic_reco d,
      forward      : ARRAY [0..5] OF four_byte_statistic_reco d,
      next_rec     : ↑ip_network_rec,
   RECEND;
```

Creation/Modification

1. The main structure of the SDS will be allocated upon initialization of the IP module.

2. The individual network records will be created by the 3A

status processor.

3. When the buffer is changed all active networks will be copied from the old buffer to the new buffer.

4.2.4 TIMER LIST

The timer list is a linked list of records. The first record marks the head of the list and is not used for data. The list is ordered by the time duration for each entry. Each entry in the list contains a time value which represents the interval of time which must expire after the preceeding record is removed from the list. This format requires more time when adding and deleting entries, but is very simple to process when checking for expired timers.

CYBIL data definitions

```
TYPE
  ip_timer_rec = RECORD
      time_delta : INTEGER,
      next_rec   : ↑ip_timer_rec,
      buffer_ptr : ↑reassembly_buffer,
    RECEND;

VAR
    timer_list : ↑timer_rec;
```

Creation/Modification

1. The first (base) record of the list is created by the initialization routine and always exists.

2. When each reassembly buffer is created, a timer record will be created.

3. As each datagram is processed the timer record may be updated.

4. When a reassembly buffer is deleted the corresponding timer record will also be deleted.

4.3 INITIALIZATION

The IP module will initialize itself when the first SAP is opened. A static variable will be used by the open_sap routine to determine if it is being called for the first time, and if so then the data structures will be initialized, a SAP will be opened with the statistics processor, and a SAP will be opened with the 3A intranet layer.

4.4 DESIGN CRITERIA AND ALTERNATIVES

The IP module will process each datagram that comes from and goes to an IP network. It is therefore important that the IP module process each datagram as quickly as possible and use the smallest amount of resources possible.

The IP module does not run as a separate task because of the (hopefully) small amount of time it will use to process each request, and the need to avoid the overhead of intertask messages often used for intertask communication.

One possible sacrifice of efficiency is the separation of the routing tasks into a separate module. This is a good logical grouping of functions, but could add additional procedure calls not otherwise needed.